Inside the Linux TCP/IP Stack

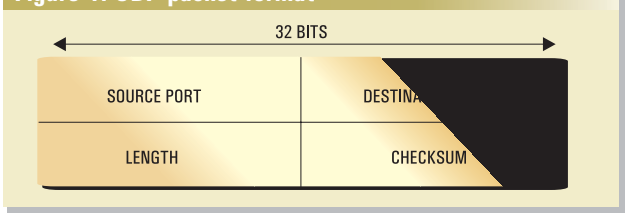
We know the Linux architecture and how the Linux TCP/IP stack is organised. Now let's see exactly what path a network packet takes from the Network Interface Card to the user.

In this article, we'll explore the interesting, though difficult path a network packet takes, when delivered from the Network Interface Card (NIC) to the user, by the Linux kernel. To keep this article simple, I have taken the UDP (User Datagram Protocol) packet as an example, but the concept applies equally well to other types of network packets.

UDP is a connectionless transport layer protocol in the TCP/IP (Transmission Control Protocol/Internet Protocol) stack. It is a simple protocol that exchanges datagrams without acknowledgements or guaranteed delivery, requiring that error processing and retransmission be handled by other protocols. UDP is defined in RFC 768 (http://ietf.org/rfc/rfc0768.txt?number=768). Figure 1 shows the packet format of UDP.

Like the modem, the NIC allows your computer to communicate with the outer world. It is a hardware card to which you connect your network cable, and its main job is to receive the packets from the network cable and pass it on to the operating system for further processing. The NIC can operate in two different modes: promiscuous and non-promiscuous. In the promiscuous mode, it will accept all the packets that it sees on the network cable (even the ones that are not destined to your host), and pass them on to the operating system. All packet capture tools like Ethereal, Snort, etc, use this technology to sniff packets from the network. In the non-promiscuous mode, it accepts only the packets that are destined to your host.

Figure 1: UDP packet format



It is the device driver software that enables the communication between the NIC and the kernel. A device driver is a software program that runs in the kernel mode. It can be either statically linked to the Linux kernel or dynamically loaded at run time.

Figure 2 shows a higher-level view of the path that a UDP packet will take from the NIC to the user space in Linux.

NIC interrupt handler

Whenever the NIC receives a network packet, the interrupt handler function of the device driver software gets called to process and pass the packet to the Linux TCP/IP stack. The interrupt handler first retrieves the network packet from the NIC memory to the system memory through DMA. It then validates the raw packet size against the underlying MTU. The interrupt handler will then try to allocate a new Socket Buffer (skb) to hold the packet. If the skb allocation succeeds, the raw

Figure 2: UDP packet processing

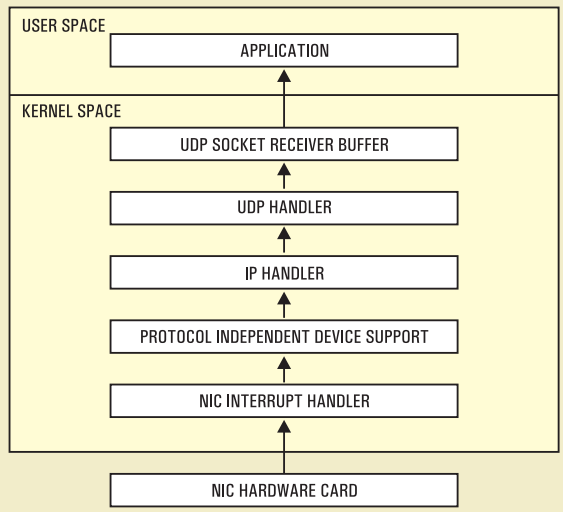
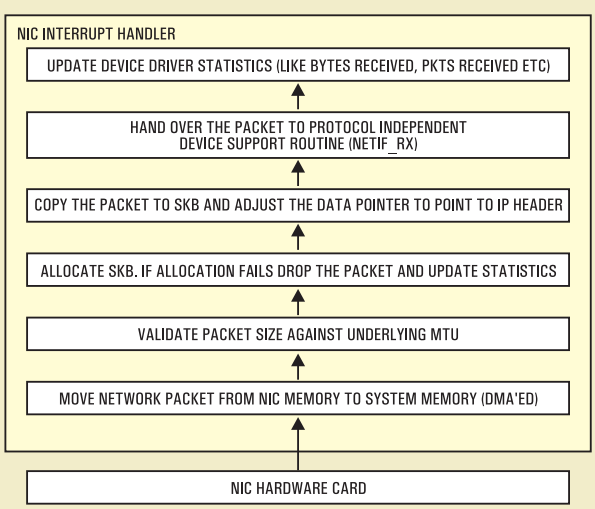


Figure 3: Packet reception at NIC

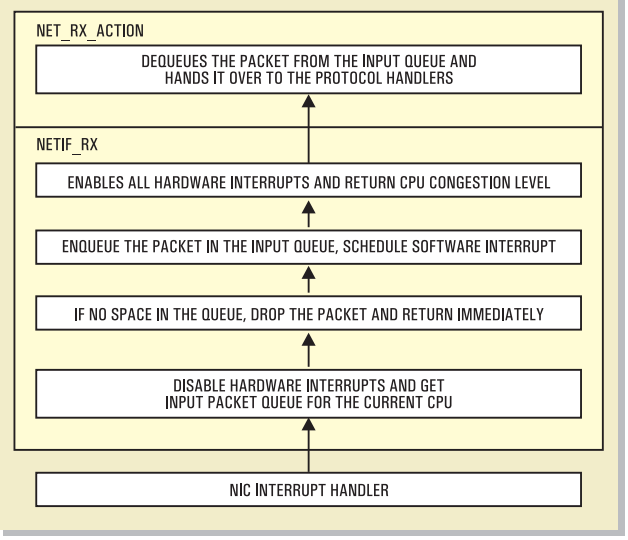


packet is copied to the newly allocated skb and the data pointer is adjusted to point to the IP header after determining the Ethernet protocol type. The skb is then handed over to the protocol-independent device support routine (*netif_rx() defined in linux/net/core/dev.c*). On the other hand, if the skb allocation fails, the packet is dropped after logging a kernel-warning message, and then the device statistics are updated accordingly. The */proc/net/dev* reports this drop. Upon returning from the protocol-independent device support routine (*netif_rx()*), the driver statistics (like bytes received and packets received) are updated accordingly. The */proc/net/dev* reports these statistics. Figure 3 illustrates this flow.

Protocol independent handler

The protocol independent handler *netif_rx* defined in *linux/net/core/dev.c* receives the network packet from the NIC interrupt handler, and queues it for the upper protocol layers for further processing. The handler disables all hardware interrupts and runs inside the NIC handler, and schedules a software interrupt handler NET_RX_SOFTIRQ (*net_rx_action* defined in *linux/net/core/dev.c*) to handle the other time-consuming tasks. In order to queue the packet, the *netif_rx* first gets the input packet queue for the current processor (the CPU on which the code is running). If there is not enough space in the queue (maximum queue length is determined by the *netdev_max_backlog*), the packet skb is freed and the packet is dropped. If n*etif_rx* is able to successfully queue the packet, then it returns the CPU congestion level to the NIC interrupt handler.

Figure 4: Inside the protocol-independent handler



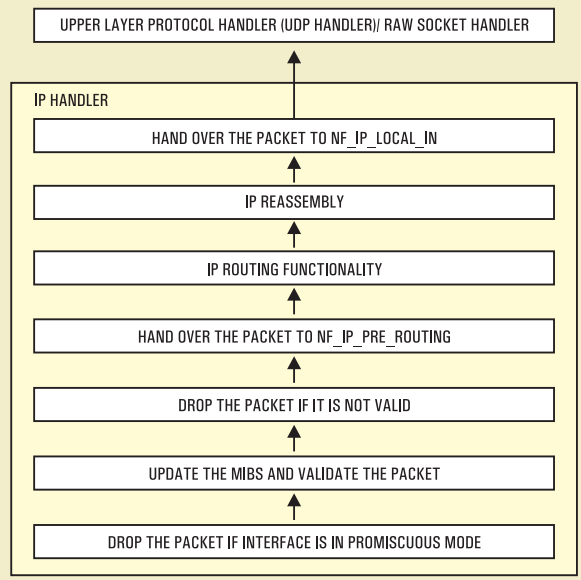
The softirq *net_rx_action* de-queues the packet from the input packet queue of the current CPU and hands it over to the protocol handlers for further processing. Figure 4 illustrates this.

IP handler

The main protocol handler for IP is the *ip_rcv* function defined in *linux/net/ipv4/ip_input.c*. The handler drops the packet if the interface is in the promiscuous mode. It then updates *IpInReceives* MIB and validates the IP header for length and checksum. If the packet does not adhere to IP RFC (RFC: 1122), then the packet is dropped (IpInHdrErrors MIB is updated) and skb is freed. If the datagram is valid, it is then handed over to the NF_IP_PRE_ROUTING netfilter hook.

The netfilter hook, on returning, calls the IP routing functionality (*ip_rcv_finish()* defined in *linux/net/ipv4/ip_input.c*) which does the routing job if the packet is destined for some other host and IP forwarding was enabled. *ip_rcv_finish* also processes the IP options, if any. The IP routing functionality hands over the packet to *ip_local_deliver* defined in *linux/net/ipv4/ip_input.c* if the packet is destined for the current host. It also reassembles the packet if the packet was fragmented.

Figure 5: Inside IP handler



The *ip_local_deliver* then calls the NF_IP_LOCAL_IN netfilter hook, which on returning, calls the *ip_local_deliver_finish* defined in *linux/net/ipv4/ip_input.c* to hand over the packet to the next higher level protocol handler (*udp_rcv* incase of UDP packet). If any raw socket is opened, then the packet is also handed over to the raw socket handler.

'ICMP destination unreachable or protocol unreachable' is returned, if *ip_local_deliver_finish* cannot find any suitable protocol handlers for the packet.

UDP handler

The main protocol handler for handling UDP packets is *udp_rcv* defined in *linux/net/ipv4/udp.c*. The *udp_rcv* first updates the IpInDelivers MIB and validates the packet header and checksum. If the packet is not valid, then it updates UdpInErrors MIB, and skb is freed and the packet is dropped. If the packet is valid, then it looks for any open matching UDP socket. If it finds one, then the packet is queued in the socket receive buffer to be consumed by user space applications. If there are no open UDP sockets, then the MIB UdpNoPorts is updated and the 'ICMP destination unreachable or port unreachable' is returned.

Well, that's it! The UDP packet has almost reached the user. It is now all in the hands of the user space application to process and respond. I hope this article will help you explore the Linux TCP/IP stack with more clarity than before. The new insight into the areas of performance tuning, implementing network device drivers, user space stack implementation, bypassing the stack for high performance, etc should allow you some happy hacking!!! LFY

By: Raja R K. The author is a lead engineer working with HCL Technologies (Cisco Systems Offshore Development Centre) in Chennai. He can be contacted at: rajark_hcl@yahoo.co.in